Plaquage omnidirectionnel de textures provenant de séquences vidéo

Alexandrina Orzan, Jean-Marc Hasenfratz

Journées de l'Association Francophone d'Informatique Graphique, Bordeaux, 2006

Plaquage omnidirectionnel de textures provenant de séquences vidéo

Alexandrina Orzan [1] et Jean-Marc Hasenfratz[1,2]

[1]ARTIS/GRAVIR-IMAG INRIA
[2]Université Pierre Mendès-France

Abstract

Nous proposons dans cet article d'identifier clairement les problèmes rencontrés lorsque l'on veut effectuer un texturage omnidirectionnel d'un modèle 3D. Nous proposons aussi des solutions à chacun de ces problèmes. Une version anglaise, plus détaillée, se trouve en deuxième partie.

1. Introduction

De plus en plus de recherches sont effectuées dans le domaine de la vidéo 3D. L'approche la plus commune est l'utilisation de plusieurs caméras fixes filmant une même scène. Le but est alors d'obtenir une vidéo *free-viewpoint*, où l'utilisateur peut choisir son point de vue pour visualiser la scène. Le choix est interactif et sans limitation de position.

Les applications possibles sont diverses. Un système *free-viewpoint* peut augmenter le réalisme dans le cadre de téléprésence. De ce fait, des utilisateurs situés physiquement en différents endroits peuvent collaborer à travers un même environnement virtuel. D'autre application concernent les effets spéciaux employés par l'industrie cinématographique, comme ceux introduits dans le film Matrix (*freeze-and-rotate*). Ils seraient alors rendus accessibles à tout utilisateurs.

Pour les vidéos de type *free-viewpoint*, la scène est filmée simultanément par différentes caméras depuis plusieurs points de vue. Les flux vidéo obtenus par les caméras sont utilisés pour créer un modèle 3D de la scène. Cette reconstruction tridimensionnelle est indispensable pour que l'utilisateur puisse regarder la scène depuis n'importe quel point de vue. Dans le cadre de la réalité virtuelle, il est possible d'ajouter de nouveaux objets dans cette scène (objets virtuels) et de traiter les problèmes d'éclairage (ombres au sol…), ainsi que les problèmes d'occultation [HLGB03, HLS04].
Pour rendre le modèle plus réaliste, les flux vidéo provenant des caméras sont plaqués sur le modèle 3D. En combinant le modèle 3D reconstruit et les différents flux vidéo, nous sommes ainsi capables de reconstruire un monde virtuel réaliste.

2. Identification des problèmes.

La reconstruction 3D utilisée dans cet article est une reconstruction *model-free*, pour laquelle nous ne disposons d'aucune information concernant l'objet reconstruit. La méthode utilisée permet de traiter des scènes dynamiques et complexes en temps réel. Cependant, la reconstruction fournie n'est pas parfaite et entraîne un certain nombre d'artéfacts lors du texturage du modèle.

Visibilité

Nous distinguons trois types de problèmes liés à la visibilité. Le premier concerne l'auto-occlusion de certaines régions. En effet, une partie du modèle peut être caché par une autre. Il faut alors déterminer les "bonnes" caméras à utiliser lors du plaquage des textures sur cette région. Le deuxième problème que nous classons dans la "visibilité" concerne les zones qui sont vues par une ou plusieurs cameras sous un angle proche de 90°, pour lesquelles l'information de texture est peu fiable. Enfin, notre troisième catégorie regroupe les parties du modèle qui ne sont vues par aucune caméra. Cette absence de visibilité peut être très courte (quelques images) dans le cas de problème de précision de la capture ou beaucoup plus longue dans le cas d'un manque et/ou d'une mauvaise disposition des caméras filmant la scènes.

Couleur des flux vidéo

Nous disposons d'un ensemble de flux vidéo provenant des différentes caméras. Certain points du modèle peuvent donc

être filmés sous différents angles. Lors du rendu final de l'image du modèle, il est nécessaire de "mélanger" les flux pour éviter de trop fortes discontinuités. Ce "mélange" peut entraîner différents problèmes :

- La couleur d'un objet est perçue différemment pour différentes directions, à cause de la réflexion spéculaire.
- Dans les images filmées, un pixel va représenter une quantité de la surface différente selon la distance de l'objet et l'angle de la vue.
- Il se peut que des erreurs de calibrage apparaissent.

Qualité du modèle
La reconstruction 3D n'est pas toujours conforme à la structure réelle de la scène. Ceci est particulièrement le cas avec une approche de type *carving*. Ces différences géométriques entre la vraie scène et celle reconstruite donnent lieu à des plaquages de texture erronés.

Cohérence temporelle
Dans les approches *model-free*, le modèle est reconstruit à chaque image sans tenir compte des reconstructions précédentes. En particulier, le nombre et la forme des polygones changent à chaque reconstruction et donc pour chaque image. Le problème est alors d'assurer une cohérence temporelle, de sorte que les couleurs demeurent les mêmes d'une image à l'autre pour chaque partie du modèle.

Echantillonage
Comme dans les approches de type *free-viewpoint* l'idée principale est de permettre à l'utilisateur de bouger librement dans la scene récréee, le *resampling* pose un problème pour le texturage, aussi bien la *pixelization* (lorsque l'on zoome sur le modèle) et l'*aliasing* (lorsque l'on s'éloigne du modèle).

3. Résolution des problèmes

Une fois ces problèmes identifiés, nous avons essayé d'y apporter des solutions.

Pour résoudre les problèmes de **visibilité**, nous avons employé, dans une première passe, la technique de *shadow-mapping*. Cette méthode nous permet de déterminer, pour chaque caméra, les points cachés par d'autres objets, mais elle n'est pas fiable pour les surfaces tangentes à la direction de vue des cameras. C'est pour ceci que l'on considère cette surface comme invisible.

Pour essayer de gommer les **imperfections du modèle**, nous utilisons plusieurs techniques:

- Nous considérons comme invisibles les frontières de silhouette, car elles appartiennent souvent au fond et non au modèle.
- Nous identifions les parties de la scène où il y a une grande différence de profondeur. Dans ces parties, il est très probable que le modèle présente des imperfections. Nous utilisons alors des textures provenant d'autres caméras.
- Quand au mois trois caméras voient un point du modèle, nous utilisons l'écart type dans l'espace couleur HSV afin d'éliminer les couleurs erronées. Cette méthode nous permet de repérer aussi les différences de couleurs provenant de la réflexion spéculaire. Quand nous disposons de deux caméras, la même approche nous permet de vérifier si les couleurs sont proches ou non.

Une fois que nous avons décidé quelles sont les couleurs qui peuvent être mélangées, nous définissons une fonction de *blending*, pour avoir une transition douce entre les différentes textures. Cette fonction est basée sur la variation de l'intensité lumineuse qui arrive sur l'objet depuis chaque camera.

Puis, nous procédons à la **correction d'erreurs**. Nous essayons de remplir les portions invisibles ou pour lesquelles les cameras ne se sont pas "mises d'accord". Pour ces parties nous appliquons un filtre médian sur les plus proches voisins déjà texturés. Cette variation de l'algorithme nous permet de texturer sans avoir à "introduire" des nouvelles couleurs, mais en gardant la couleur dominante dans le voisinage.

4. Conclusion

En conclusion, nous avons identifié et présenté en détail les problèmes du texturage omnidirectionnel. Nous avons proposé un algorithme pour résoudre certain d'entre eux.

Nous avons employé la technique de *shadow-mapping*, avec des améliorations, pour déterminer au mieux les parties visibles du modèle 3D. Nous avons pris en compte les imperfections du modèle et nous avons défini une méthode pour identifier les "zones de risque". Nous avons proposé d'éliminer les fausses couleurs en faisant un test d'écart type. Nous avons utilisé le filtre médian pour remplir les "trous".

Abstract

In video-based rendering, real dynamic scenes are captured by video cameras and replayed, so that they can be seen by the observer from any viewpoint. In order to permit a complete immersion of the real scene in a virtual environment, some approaches reconstruct a 3D shape and then textured it with images taken from the cameras. This paper proposes to identify the problems of multi-view texturing for video-based rendering and to offer solutions for a number of them. We successfully treat visibility issues, identify "risk zones", correct projection displacement errors and fill in small untextured areas. Our algorithm works in real time, thus permitting an interactive viewing of the augmented scene.

1. Introduction

Video-based rendering [Mag05] aims to capture the 3D appearance of the real-world and to permit the user to watch the filmed scene from an interactively chosen point of view.

There are several ways of creating free-viewpoint video, but in order to ensure a realistic appearance, all have to deal with reproducing the "coloring" of the real scene. We concentrate on texturing a 3D model produced that provide explicit reconstruction of the geometrical model and don't make any assumption on the observed scene. These methods are discussed in section 2.

Given the 3D model, we attempt to identify, in section 3, the problems of texturing it from multiple view video sequences and to explain the origin of these problems.

Section 4 contains a review of related work.

We then present, in section 5, our approach to the omnidirectional texturing problem. We describe an algorithm that deals with visibility problems, identifies errors produced by model imperfections, introduces a blending function and textures invisible areas.

Section 6 discusses our results, while section 7 ends this article with the conclusions and future tasks.

The main contribution of this paper is that it proposes a framework for omnidirectional texturing. We can thus describe a real-time algorithm that treats several clearly defined problems, such as visibility leaks, texture displacement and small invisible areas.

2. Context and Motivation

The traditional way of capturing real scenes is by using image-based rendering (IBR) techniques, which render novel views of the scene from input photographs [SK00]. For dynamic scenes, we do this by filming the scene with multiple video cameras and using the video sequences as input. Depending on how much geometric information is recovered, the IBR techniques can be classified in three categories: rendering without geometry, rendering with implicit geometry [LMS04b] and rendering with explicit geometry [HLS04].

We focus on methods that offer an explicit *3D reconstruction* of the real scene, because our purpose is not simply to anticipate how the scene would look from an arbitrary point of view, but to permit a complete immersion of real actors and objects in a virtual environment. This means that interaction between real persons and virtual objects is made possible and that the real scene can be lighted by virtual lights and cast shadows on virtual objects. In order to do so, we need to know the 3D geometry [HLGB03, HLS04].

Next, out of two possible approaches, model-free and model-based, we prefer the *model free* techniques, meaning that we don't have any a priori knowledge on the reconstructed scene. The reason for this is that we would like to be able to reconstruct several actors and objects, not just a human, as in [TCMS04, HS03]. Moreover, model-based approaches impose certain constrains on the reconstructed object, such as close-fitting clothes for a human.

Even more important, in order to have interaction between the actors and the virtual environment we need an *end-to-end real-time* system. By using model-based techniques, this is difficult to obtain, because of the "fitting" process between the model and observed data.

To make the model appear closer to reality, we map images captured from the video streams onto the 3D shape. Texturing the model has the advantage of registering small real-life details, such as cloth creases, thus making the relatively simple surface seem more complex.

3. Problems of omnidirectional texturing

In this section we describe the problems encountered when mapping multiple textures on a 3D model reconstructed with a model-free technique from video streams.

Although some of the problems presented here also appear when texturing a still model or a model-based reconstruction, there is only a partial overlap. In the first case, because we deal with a dynamic input (vs. a still one), and in the second because we do not have the implicit knowledge of temporal coherence that the model-based techniques have.

What one wants to achieve in omnidirectional texturing is:

- colors close to the real ones for the entire surface;

- spatial smoothness, meaning that the texture should be coherent all around the model;
- temporal smoothness, which implies that the 3D model parts should conserve the same color over time.

3.1. Color difference between camera images

In omnidirectional texturing, several images are used to texture an object. In doing so, areas of the 3D model will be mapped with more than one texture, and the necessity of blending colors from different camera images appears. This mixing of colors masks the lack of photometric agreement in source images and the projection displacements caused by an imperfect model.

However, blending has the downside of causing blurring, thus ruining the "crisp" input images. Moreover, the color difference will still be noticeable if the blending is not done smoothly across the surface.

Concerning the camera image disparities, the colors associated with the same 3D point might not be the same for several reasons:

- The object color is indeed different for different directions, due to specular reflection.
- The image pixels cover a different "amount" of the surface, depending on the distance from the object and on the angle of view.
- Errors appear in camera calibration.
- Noise is apparent over time for each individual camera.

For the first point, the assumption is made that the reconstructed surfaces are (nearly) Lambertian, but this is not always the case. For surfaces that have a specular reflection component, the viewing position influences the amount of light perceived, and so the cameras register dissimilar colors.

Next, a pixel in camera images can correspond to a smaller of bigger area on the real surface, depending on whether the cameras are closer or farther away from the object. The pixel color is thus composed by a "set" of rays that vary in number from camera to camera. As explained in Figure 1, this may result in one camera seeing violet when another sees red and blue.

Lastly, color disparity may also originate from errors in white-balancing the cameras and noise. This fact makes images difficult to superpose even when taken from an approximately equal distance.

3.2. Model Quality

The second major source of errors consists in wrongly texturing the virtual model because of the geometrical disparities between the real scene and the reconstructed one.

When the model doesn't conform closely to the actual

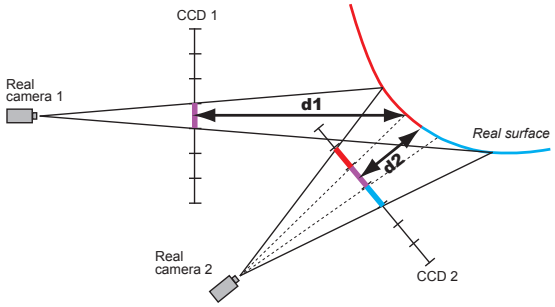

Figure 1: *Camera image pixels can represent different "amount" of surface.*

structure of the scene, projecting the image of the real object on the 3D reconstruction will cause projection displacements. A line seen by two cameras might appear as two curves of opposite curvature, for example. A schema explaining this is shown in Figure 2 and a concrete example can be seen in Figure 3.

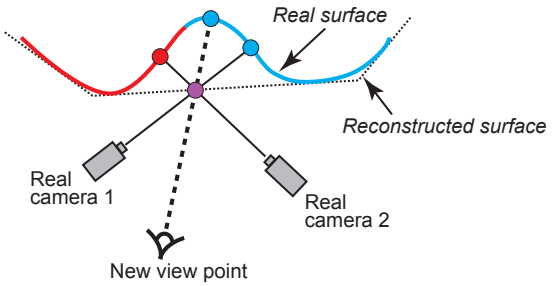

Figure 2: *Texture projection can have errors due to geometric disparities between the real and the reconstructed model.*

3.3. Visibility

The visibility problem consists in correctly determining what parts of the reconstructed model are visible from each camera. This is important because reference view images should be used to texture only those portions of the model that the camera actually sees.

Moreover, the visibility test is performed on the imperfect model, so areas considered visible might actually not be visible in the real scene. This will lead to "visibility leaks", and therefore to wrong textures, in self-occlusion areas.

There is also the problem of "badly visible" areas. These are the regions that, although seen by the camera, are viewed under an angle close to 90°. Therefore, very little texture information is obtained, and that is deformed and unreliable. This is the case for areas close to silhouette edges.

Invisible portions of the model also pose a problem in

multi-view texturing, since we are lacking the information needed to assign color to them.

3.4. Resampling

The main idea of free-viewpoint video is to allow the user to move freely in the recreated scene. This means that he is able to get as close or as far as he wants to the textured 3D model.

However, the camera images used to texture the model are of a fixed resolution, and generally less then what is needed for a monitor screen. Thus, when the textured object is zoomed in or out, visually disturbing artifacts will appear. Especially, as we get closer, the pixelization will become noticeable. Furthermore, errors and blurriness due to blending are more visible when zooming in.

Zooming out can also lead to problems, such as aliasing.

3.5. Temporal coherence

For model-free reconstruction, there is usually no intrinsic temporal coherence of the 3D object. In this case, the structure of the scene is not consistent over time, a fact visible when the reconstruction is seen as a sequence.

When multiple textures are applied to this non-coherent model, there is no guaranty that the resulting color will only change coherently in time. Moreover, if texture blending is based on model characteristics, such as normals, the color is likely to change every frame.

The color variations cause a disturbing effect of flickering, especially visible when there are flagrant texture errors appearing and disappearing with each separate frame.

4. Related work

Three-dimensional production from multiple view video, or 3D video, was first popularized by Kanade et. al. [KRN97, NRK98], who proposed to obtained an immersive visual medium that lets the viewer select his viewing position. Kanade named this medium *Virtualized Reality*, since it *virtualizes* the event in order to permit the free movement of the user. [VBK05]

Debevec et al. [DTM96, DYB98] put forth a view depending texture mapping approach (Façade). While their approach guarantees smooth transition between views, it does not guarantee smooth transition between weights across the surface, so seams will appear.

In Buehler et al. [BBM*01], the core idea is that the influence of a single image on the final rendering is a smoothly varying function across the desired image plane (or, equivalently, across the geometry representing the scene). The paper introduces a resolution penalty and proposes a formula for blending weight based on a linear hat function. But the algorithm does not explicitly handle problems of visibility and when they add a visibility treatment [MBM01], the *blending field*is not always smooth.

[LMS03] introduces the idea of using shadow maps in order to solve visibility problems at a point level, rather than triangle level, as before. It also encodes an alpha value in the silhouette images; this value is the result of a morphological erosion applied on the silhouette and is incorporated in the blending weight.

[LMS04a] proposes an algorithm that makes use of the graphics hardware. [CTMS03] does a *model-based* reconstruction of a human actor and proposes that one layer of boundary pixels in each silhouette should be removed and filled with adjacent foreground pixels.

The research in video-based rendering was gathered in a book by Marcus Magnor [Mag05] and were subject to a Siggraph 2005 course.

5. Practical example

We present an omnidirectional texturing algorithm that maps multiple textures on a dynamic 3D model. The input for our algorithm is the geometrical model of a real human, reconstructed frame per frame, together with the camera space positions and video streams. The 3D model we used is recovered in real time through a model-free reconstruction method, described in [BF03].

5.1. Visibility

As in [LMS04a], we use shadow maps as a first "method" in deciding which parts of the model are visible from which camera. This method has several advantages: it is a general method for computing shadows, meaning that it works for all types object definition (voxels, polygons). It is also a very fast method, that can be implemented completely on the GPU. This is a very important consideration, since we have to manipulate a set of 9 cameras, and this for each frame.

However, shadow mapping is prone to errors when zooming into shadow boundaries (perspective aliasing). The method is also sensible to projective aliasing. This means that when the light rays "departing" from the camera point of view are almost parallel to a surface, the shadow stretches along that surface and the lighted and shadow areas are not clearly distinguished..

Moreover, the almost parallel surfaces are badly visible from the camera and the texture information is not reliable. We therefore consider invisible all points P for which $\vec{N} \cdot \overrightarrow{C_i P}$ is close to 0, where $\vec{N}$ is the normal to the surface at P and C_i is the optical center of camera i.

Since shadow mapping is based on storing depth information in textures, we need to restrain the number of cameras

we use in texturing at the number of TEXCOORD parameters defined in Cg. We therefore have to have a quick method of deciding which of the 9 cameras should be used at each time frame.

To do this, we consider the center of the scene to be the center of the model's bounding box and we compute the angle with which each camera deviates from the current point. We define the penalty $penalty(C_i, V)$ as being the angular deviation of camera center C_i from the current viewpoint V. We then consider only the first 6 cameras having the smallest penalty.

5.2. Model /real scene disparities

The big problem in texturing a reconstructed model is that what the cameras see is not what you get as a 3D model. Errors intervene in the reconstruction chain and the result is only an approximation.

It is then necessary to identify the 3D model areas more likely to have been subjected to error. These areas we call "risk areas".

One source of erroneous colors in texturing comes from the background / foreground separation done before reconstructing the model. Pixels that are on the frontier between foreground and background are considering as belonging to the foreground, but their color actually retains color information from both, giving place to false color information.

For this reason, we opt to remove the texture information contained by silhouette frontier. We do this by introducing an alpha component to the texture, at the loading moment. The frontier pixels will have decreasing alpha values, to signal that the color is not trust-worthy. Thus, when the texture is mapped on the object, we can choose to disregard the color.

Another problem in texturing an imperfect model is that the separation between parts with different depths is not always conform with the real case. Thus, a hand can often project on a torso, for example.

We propose a method to eliminate this cases which we call "border removal". The idea is that this situations arrive in the portions that are at the border between shadow and light for one particular camera. We use a method described in [CD04] to identify the pixels on the frontier. This method is based on the fact that deciding if a pixel is in the shadow (value 0) or not (value 1) actually involves computing an average on four neighbors. Thus, a border pixel value will be neither 0, nor 1, and can be identified by two comparisons.

Until now we concentrated on the problems of mapping a single camera image on the model. But once several images are projected on the 3D reconstructing, another problem becomes visible. This is the projection displacements problem.

As explained in Section 3.2, projecting on a imperfect model leads to texture displacements, such that, for e.g., the eye viewed from one camera will not project in the same position as the eye from another camera image. Mixing this different projections leads to blurry and incorrect results.

For this, we tried to choose which color should be the final one. We therefore introduced a standard deviation test in HSV color space. Passing from RGB space to another space is necessary, because RGB doesn't encode the hue (the property of a color that varies from red to green). We should be able to say that orange is closer to red than blue. These relationships are reflected in HSV, a color space that has the added advantage of being easily computed from RGB.

We therefore pass to the HSV values from a RGB color, and then consider only the H (hue) for the standard deviation. If a sufficient number of color values are available for a pixel (at least 3), we compute the mean (μ) and the standard deviation (τ) for H channel. Individual colors falling outside the range $\mu \pm \beta \cdot \tau$ are excluded. The factor β permits us to modify the confidence interval for which the colors are accepted.

In the case of only 2 available colors, we introduce a measure of trust, and if the standard deviation surpasses it, we classify the area as not trust-worthy and we postpone a decision on the color until the "error correction" phase.

5.3. Color blending

Once the trusted colors for each pixel of the rasterized scene are settled upon, we need to blend them in a seamless way.

This means that a function that varies smoothly across the camera image should be found. But the visibility tests and the color corrections applied in our algorithm, combined with this function, would have as a result a discontinuous function (continuity on the border ruined).

In order to preserve the smoothness of the blending function, while eliminating badly visible areas, we tie the variation not to the camera image, but to the model. Thus, we interpret the camera as a pointlight, and compute the variation of the light intensity on the model: $lightIntensity_i = max(cos(\vec{N}, \overrightarrow{C_iP}), 0)$.

Then, we modify cos curve to consider the eliminated tangent areas

$$lightIntensity[i] = (a \cdot pow(lightIntensity[i], 2)) + b \cdot lightIntensity[i] + c$$

where $a, b$and c where chosen such as to bring the lightIntensity value to 0 then it nears a threshold value.

We define the blending weight of a camera C_i for a point P as:

$$weight(C_i, P) = visible[i] \cdot notSilhouetteFrontier[i] \cdot notBorder[i] \cdot lightIntensity[i] \cdot cameraEyeAngle[i]$$

where $cameraEyeAngle[i]$ is the cosinus of the angle between the eye and the camera center C_i. This term ensures the smooth "apparition" and "disparition" of a camera.

Because the blending maximum value is 1, the weight function is summed to 1 by the formula:

$$\overline{weight}(C_i,P) = \frac{weight(C_i,P)}{\sum_{j=1}^{n} weight(C_j,P)}$$

where n is the number of cameras

5.4. Error correction

In a second pass over the computed texture, we try to fill in the areas that where invisible to all cameras or on which the camera colors "didn't agree".

For this areas, we use a median filter on the closest visible neighbors. The median filter was chosen because its main property is that it eliminates outliers without creating new pixels values. Thus, we color the invisible areas without having to introduce new color values, by preserving the dominant color.

6. Results

Our results were obtained with the 9 camera system presented in Figure4. The 3D model we used was a polyhedral model of approximately 5000 polygons.

The algorithm was implemented using OpenGL and CG; a parallel thread was developed that allows us to load the camera images and render the final texture at the same time.

We tested our implementation on a computer having an Intel 2.40GHz CPU and a GeForce 6800 GT graphic card. For a rendered novel view of a 780x582 resolution, our algorithm reaches 30 fps, thus succeeding in loading the video images and rendering the textured scene in real time.

Examples of results can be seen in Figures 3 and 4

7. Conclusions and future work

We identified and presented in detail the problems that rise in omnidirectional texturing. We then proposed an algorithm that solves some of these problems.

We showed a way of better determining which are the model parts truly seen by the video cameras. We considered the model imperfections and we defined a method to identify which are the "risk areas". We also defined a blending function that accounts for these areas and makes use of the smoothly varying normals to preserve continuity.

We made use of the standard deviation in order to eliminate the erroneous colors and the invisible parts were filled in by the aid of the median filter.

Also, our implementation is appropriate for any type of model (be it a voxel or a polyhedral model), as long as a normal to the surface is provided or can be computed.

Still, we have not addressed the problem of resampling. Related to it, the shadow mapping problem of perspective aliasing remains untouched. One solution would be to use perspective shadow maps [SD02].

Another unsolved problem is that of temporal coherence. While the video images are coherent, our system of blending them does not guaranty that the result will also be temporal coherent. A way of maintaining each camera contribution over time needs to be found.

Acknowledgments

Alexandrina Orzan is supported by a grant from the European Community under the Marie-Curie project VISITOR MEST-CT-2004-008270.

References

[BBM*01] BUEHLER C., BOSSE M., MCMILLAN L., GORTLER S., COHEN M.: Unstructured lumigraph rendering. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM Press, pp. 425–432.

[BF03] BOYER E., FRANCO J.-S.: A hybrid approach for computing visual hulls of complex objects. In *CVPR (1)* (2003), pp. 695–701.

[CD04] CHAN E., DURAND F.: An efficient hybrid shadow rendering algorithm. In *Proceedings of the Eurographics Symposium on Rendering* (2004), Eurographics Association, pp. 185–195.

[CTMS03] CARRANZA J., THEOBALT C., MAGNOR M., SEIDEL H.-P.: Free-viewpoint video of human actors. *"ACM Trans. on Computer Graphics 22*, 3 (jul 2003).

[DTM96] DEBEVEC P. E., TAYLOR C. J., MALIK J.: Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. *Computer Graphics 30*, Annual Conference Series (1996), 11–20.

[DYB98] DEBEVEC P. E., YU Y., BORSHUKOV G. D.: Efficient view-dependent image-based rendering with projective texture-mapping. In *9th Eurographics Workshop on Rendering* (1998).

[HLGB03] HASENFRATZ J.-M., LAPIERRE M., GASCUEL J.-D., BOYER E.: Real-time capture, reconstruction and insertion into virtual world of human actors. In *Vision, Video and Graphics* (2003), Eurographics, Elsevier, pp. 49–56.

[HLS04] HASENFRATZ J.-M., LAPIERRE M., SILLION F.: A real-time system for full body interaction. *"Virtual Environments"* (2004), 147–156.

[HS03] HILTON A., STARCK J.: Model-based multiple view reconstruction of people. In *IEEE International Conference on Computer Vision* (2003), pp. 915–922.

[KRN97] KANADE T., RANDER P., NARAYANAN P. J.: Virtualized reality: Constructing virtual worlds from real scenes. *IEEE MultiMedia 4*, 1 (1997), 34–47.

[LMS03] LI M., MAGNOR M., SEIDEL H.-P.: Improved hardware-accelerated visual hull rendering. *Proc. Vision, Modeling, and Visualization* (Nov. 2003), 151–158.

[LMS04a] LI M., MAGNOR M., SEIDEL H.-P.: A hybrid hardware-accelerated algorithm for high quality rendering of visual hulls. In *GI '04: Proceedings of the 2004 conference on Graphics interface* (School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004), Canadian Human-Computer Communications Society, pp. 41–48.

[LMS04b] LI M., MAGNOR M. A., SEIDEL H.-P.: Hardware-accelerated rendering of photo hulls. *"Computer Graphics Forum" 23*, 3 (2004), 635–642.

[Mag05] MAGNOR M.: *Video-based rendering*, first ed. A K Peters, Ltd., 2005.

[MBM01] MATUSIK W., BUEHLER C., MCMILLAN L.: Polyhedral visual hulls for real-time rendering. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques* (London, UK, 2001), Springer-Verlag, pp. 115–126.

[NRK98] NARAYANAN P. J., RANDER P., KANADE T.: Constructing virtual worlds using dense stereo. In *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision* (Washington, DC, USA, 1998), IEEE Computer Society, pp. 3–11.

[SD02] STAMMINGER M., DRETTAKIS G.: Perspective shadow maps. In *Proceedings of ACM SIGGRAPH 2002* (July 2002), Hughes J., (Ed.), ACM Press/ ACM SIGGRAPH.

[SK00] SHUM H., KANG S. B.: Review of image-based rendering techniques. In *VCIP* (2000), pp. 2–13.

[TCMS04] THEOBALT C., CARRANZA J., MAGNOR M. A., SEIDEL H.-P.: Combining 3d flow fields with silhouette-based human motion capture for immersive video. *"Graphical Models" 66*, 6 (2004), 333–351.

[VBK05] VEDULA S., BAKER S., KANADE T.: Image-based spatio-temporal modeling and view interpolation of dynamic events. *ACM Trans. Graph. 24*, 2 (2005), 240–261.

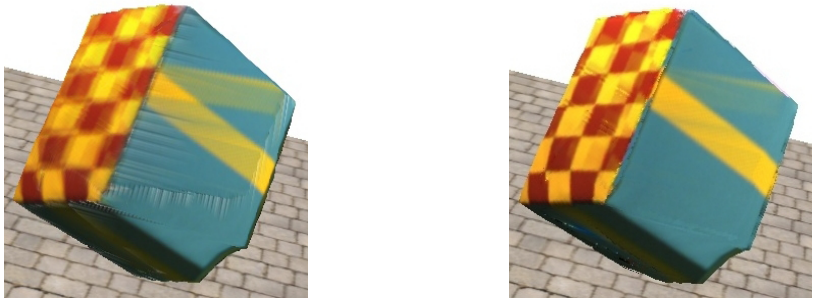

Figure 3: *Omnidirectional texturing without error correction (left) and with error correction (right)*

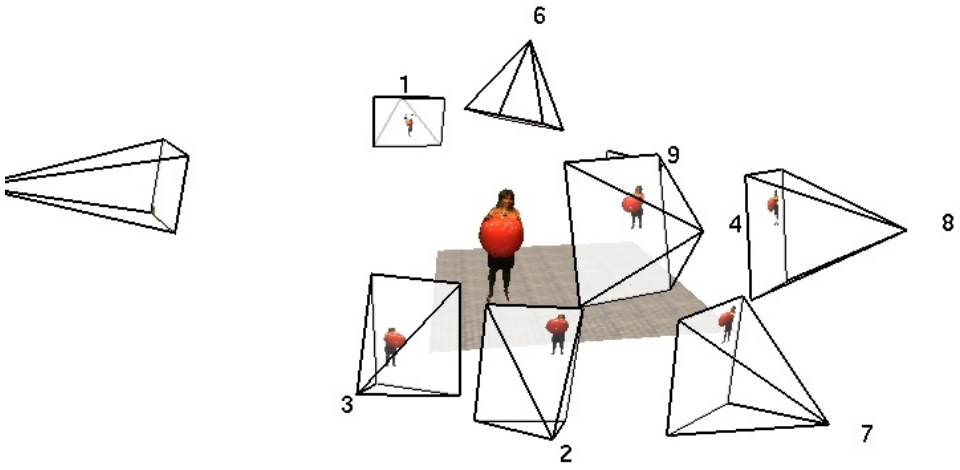

Figure 4: *The 9-camera system used for omnidirectional texturing.*